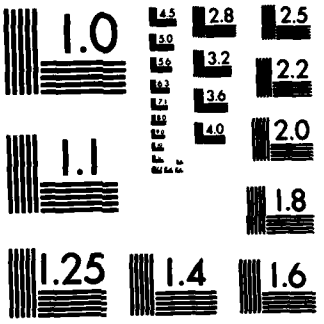
AD-A150 563 USABILITY SPECIFICATIONS AS TOOL IN ITERATIVE DEVELOPMENT(U) IBM THOMAS J WATSON RESEARCH CENTER YORKTOWN HEIGHTS NY J M CARROLL ET AL. 03 APR 84 1/1

UNCLASSIFIED RC-10437 F/G 5/1 NL



RC 10437 (#46642) 4/3/84
Computer Science/Cognition 21 pages

Research Report

USABILITY SPECIFICATIONS AS A TOOL IN ITERATIVE DEVELOPMENT*

John M. Carroll
Mary Beth Rosson

Computer Science Department
IBM Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
San Jose · Yorktown · Zurich

RC 10437 (#46642) 4/3/84
Computer Science/Cognition 21 pages

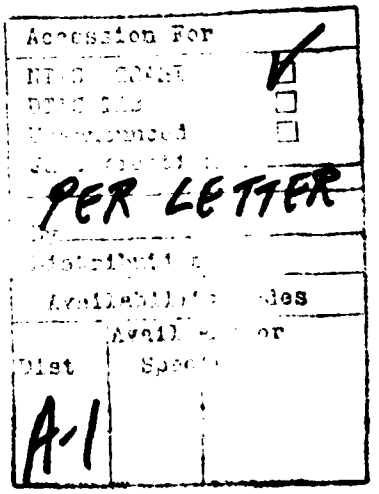
USABILITY SPECIFICATIONS AS A TOOL IN ITERATIVE DEVELOPMENT*

John M. Carroll
Mary Beth Rosson

Computer Science Department
IBM Watson Research Center
Yorktown Heights, NY 10598

Abstract: Poor usability of contemporary computer systems has impelled a concern with how usability can play a role in the system design process. In this chapter, we criticize approaches that consist of usability *analysis* of fully articulated design alternatives. Such approaches cannot in principle support the design *process* that produced the alternative designs in the first place. We develop an approach based on *usability specifications:* precise, testable statements of performance goals for typical users carrying out tasks typical of their projected use of the system. These in turn must be factored into their behavioral prerequisites, which we call *subskills,* in order to pinpoint and remedy specific problems in a design. Like functional specifications, the usability specifications and the subskills they imply are viewed as being iteratively elaborated and refined throughout the design process. An extended example from the domain of text editing is presented.

To appear in H.R. Hartson (Ed.), *Advances in Human-Computer Interaction,* Norwood, NJ: Ablex Publishing.



1. Introduction.

Computer systems are often more difficult to learn and to use than people would like them to be. This problem impacts more and more people, as computer use becomes less a matter of esoterica, and more a commonplace of daily life. The extension of computer use to "casual" users, as they are sometimes called, has also made the problem more difficult: Users cannot be expected to be engineers and scientists, to read lengthy manuals with great care or to put up with cluttered interface displays and awkward dialogues. They will not take the time or spare the effort.

Much has been said about this "usability" problem regarding current interface designs. Less has been said about how to solve the problem. In this chapter, we develop an approach to the problem based on *usability specifications:* precise, testable statements of performance goals for typical users carrying out tasks typical of their projected use of the system. These in turn must be factored into their behavioral prerequisites, which we call *subskills,* in order to pinpoint and remedy specific problems in a design. Like functional specifications, the usability specifications and the subskills they imply are viewed as being iteratively elaborated and refined throughout the design process. Since the computer system development process is already organized around the development, refinement, and implementation of specifications, this affords a means of incorporating user interface issues into the existent development process.

To investigate the role of usability concerns in the design of user interfaces, we need to examine how design is structured -- both in fact and in principle. We undertake this in four steps. First, we review some of the leading analytic ideas from design science, that is, from the theoretical study of design as a human activity. Second, we turn to an empirical investigation of design activity which questions several of the idealizations and assumptions inherent in the analytic work. In a word, the empirical work reveals design to be a process of discovery, one often marked by radical restructuring, where the analytic approaches assume design to be a mechanical and single-step decomposition into subproblems.

Third, we turn to computer system user interface development, and to the problem of systematically addressing usability concerns within this design process. We argue that current analytic approaches to usability in interface design rest upon the same idealizations and assumptions that have flawed analytic design science generally, and that these approaches fail to adequately address usability within a design *process.* Finally, we turn to our own proposal -- usability specifications: we argue that a successful approach to addressing usability design objectives within the development process consists of an iterating process of behavioral testing, usability goal-setting and analysis of prerequisite user subskills, and then more behavioral testing.

2. Analytic Studies of Design Problem Solving.

The most prominent approach to the study of design is the analytic approach: a design, or indeed the activity of design in general, is analyzed into elements and relations. A final design is a well-factored structure of elements and relations.

2.1 Alexander. Alexander (1964) presents the modern classic analysis of design problem solving. One of the principal problems Alexander dealt with is the interaction of different specifications in a design. There are no practical bounds on the number of relevant specifications in a design or on the extent to which the specifications can interact and trade-off, but there are severe limitations on the capacity of humans to process information. From this, Alexander reasoned -- as every theorist since has reasoned -- that complex designs have to involve the *decomposition* of the design into subproblems, each addressing a more specific

design subgoal (Alexander, 1964: 43 ff.). What concerned Alexander was *how* this decomposition might take place.

He distinguished between "selfconscious" and "unselfconscious" decomposition. Selfconscious decomposition is forced, *imposed* on the design. It tends to refer to a priori categorizations of design subproblems (for example, dealing with metal components separately from non-metal components on the basis of that contrast alone) or to factors arbitrary with respect to the design itself (for example, isolating large cost items from small cost items in the problem decomposition). The resultant decomposition can mistakenly construe the design problem, and therefore lead to poor design solutions. Unselfconscious design, in contrast, attempts to *discover* the decomposition that inheres in the functional organization of the design itself.

But how can this be accomplished? Alexander suggested that formal analytic methods could be developed to guide unselfconscious decomposition (e.g., 1964: Appendix 2). His approach was to represent a design as a space of interconnected points: each point is a specification, and the degree and character of its connectedness to other points models the relation between the corresponding specifications. Clusters of highly interconnected points in this representation correspond to collections of highly inter-related specifications. Alexander wanted to develop a formal method to partition the entire space of specifications into such highly interconnected clusters. An exhaustive partition developed on this basis would be an unselfconscious decomposition of the design problem, since it would be grounded entirely in the relations between specifications and not on a priori categorizations and groupings.

Unfortunately, the actual examples Alexander developed countenanced only binary relations between specifications, conflict or agreement. Thus, in his analysis of designing a vacuum cleaner, Alexander focussed on four specifications: simplicity (the number and variety of components), performance (cleaning power), jointing (ease of assembly), and economy (cost). Simplicity agrees with jointing, since a small homogeneous set of components will be easy to assemble; but disagrees, perhaps, with economy and performance. Alexander ultimately rejected his own example as over simple and idealized: specifications as general as "simplicity" will not be of much use in any real design undertaking (indeed, we will refer to specifications this general as "objectives"; for us, specifications are far more detailed statements about how an objective is to be accomplished in a design).

Alexander used the example to point out that even in a simple and idealized context, the interactions between the four specifications were numerous, varied, and complex. Nevertheless, he concluded that the formal approach of enumerating specifications and evaluating their inter-relations could be a key to understanding unselfconscious design decomposition.

2.2 Simon. Simon (1973) returned to Alexander's project a decade later and in view of many developments in modelling human problem solving behavior. He treated design problems as exemplary of a class of "ill-structured" problems, which he contrasted with well-structured problems. Simon also focussed his attention on the decomposition of a design into subproblems. Indeed, he regarded it as the single distinction between ill-structured and well-structured problems. Thus, the fundamental difference between designing a house (his example) and solving a puzzle resides in the fact that the design problem requires a decomposition. Hence Simon regards the decomposition of problem specifications as even more essential to design than did Alexander. On the other hand, Simon would consider a very large and complex puzzle to be an ill-defined problem, if its solution depended on a decomposition being made in the course of solving it.

Simon differs from Alexander in viewing "selfconscious" decomposition, in Alexander's term, as a desirable approach. He suggests a decomposition for the house design example

which imposes higher-order categories on the problem, without consideration of particular specifications (even as general as those Alexander considered) or their inter-relation:

> "HOUSE might transform to GENERAL FLOOR PLAN plus STRUCTURE, STRUCTURE to SUPPORT plus ROOFING plus SHEATHING plus UTILITIES, UTILITIES to PLUMBING plus HEATING SYSTEM plus ELECTRICAL SYSTEM, and so on."(p. 190)

This decomposition is not merely based on externally imposed a priori categories, but also is constrained by the a priori and design-external constraint that the pieces of the decomposition comprise a strict hierarchy. This indeed clarifies the distinction between Alexander and Simon on the issue of decomposition. Alexander imagines *unselfconscious* decomposition to be a matter of organizing specifications into inter-related clusters -- from the "bottom" (the level of individual specifications) "up" (the level of clusters). Simon sees decomposition as a matter of imposing a *selfconscious* categorization from the "top down".

The two analyses are very similar, however, in that they conflate "prescriptive" and "descriptive" remarks. Both accounts focus on the authors' views about what design *should* be like, but fail to seriously consider what design activity is like in fact. In both cases, this unclarity invites the unwarranted inference that successful design indeed works as the author prescribes. But in both cases only simple and idealized gedanken examples of design activity are referred to. Both analyses conceive of the decomposition of a design problem into subproblems as a single, monolithic, step. Both analyses make design appear to be a *state* instead of a *process*. Static representations are proposed, not of designs, but of design. In view of this, it seems prudent to turn briefly to some laboratory work which seeks to characterize general properties of the design process, including the *process* of decomposition.

3. An Empirical Study of Design.

Descriptive studies of the design process reveal properties rather at odds with the analyses and prescriptions of Alexander and Simon. Carroll, Thomas and Malhotra (1979) observed a design session in which a head librarian and a systems engineer tried to design better access to library computing facilities. Their description of this design activity makes several points which we will return to both in further critiquing analytic approaches to design and in developing our own proposals:

(1) Design is a *process*, it is not a state and cannot be adequately represented statically.
(2) The design process is *non-hierarchical*, neither strictly bottom-up nor strictly top-down.
(3) The process is *radically transformational*, involving the development of partial and interim solutions which may ultimately play no role in the final design.
(4) Design intrinsically involves the *discovery of new goals*.

These are not four distinct and exhaustive properties. They are more nearly four views of the same property. We separate them out to more clearly and emphatically differentiate design from other problem solving activity; we are not concerned with differentiating these properties one from another.

In the Carroll, Thomas and Malhotra library design protocol, the client begins by focussing on what might be called physical layout problems in the library, problems regarding the location and number of modems, scopes, and the local printer control unit. The interaction divides into "cycles" (see also Malhotra, Thomas, Carroll, and Miller, 1980). Each cycle is initiated by the identification of one or more design requirements and terminated by some (at least partial) treatment of these requirements.

In the early cycles of the library session, the location requirements are introduced (mainly by the librarian) and tentatively treated one by one. After several of these, the participants

pause to summarize and consolidate. While parts of a design solution have been worked out, there still seems to be too little space. At this point the idea of spooling output to a central printer is raised -- at first purely to conserve space. This leads quickly to ideas about remote access in general. Indeed, the final solution to the library problem was to develop remote access and printing capabilities.

This case study illustrates properties (1-4) and affords a more dramatic critique of the analyses of Alexander and Simon. From the very start of the session the librarian focuses on particular requirements and constraints. He does not initially organize the problem with a higher-order category such as "physical layout constraints" (Carroll et al. imposed this label in their analysis of the design protocol). Rather, he starts from the bottom. The kind of strictly top-down decomposition Simon imagines for house designing is simply not in evidence. But neither is this process much like Alexander's vacuum cleaner example. The entry level into the design process is the "bottom", but the direction of progress is not always "up".

Indeed, we found cases in which a new cycle was initiated at a higher level than the preceding one, at the same level, and at a lower level: bottom-up, bottom-over, and bottom-down. What's more, the pieces into which the design process was decomposed, the cycles, were non-accretive. Indeed, virtually all of the partial and interim solutions generated in the session were abandoned at the final design cycle when the problem was recast, essentially, in software rather than hardware terms. This involved an on-going process of goal discovery. Indeed, one way of viewing the design session is as a workshop in goal elaboration leading up to a (very brief) single design cycle.

On reflection it becomes clear that the discovery and elaboration of goals is perhaps best viewed as a defining characteristic of design-type problems. Solutions to a design problem cannot be enumerated or guaranteed a priori. This property never holds of well-structured puzzle problems, such as those in chess or theorem proving that Simon discusses. The design of a house may be constrained by various factors, but if it can be fixed completely then the design work is already finished.

A recently published journalistic account of a major computer system development project (Kidder, 1982) provides many examples of each of the design properties we have stressed in our recounting of the Carroll, Thomas, and Malhotra study. For example, the description of the design of the memory management and security architecture for the "Eagle" machine (pages 67-85) involved extended problem solving activity producing results which did not appear as any part of the final design. This suggests that the picture of design activity we have given is a general one and that the problems it raises for idealistic analytic treatments of design are general problems.

We do not mean to be design science iconoclasts. The design process, even taking properties (1-4) into account, is not inscrutable. Laboratory work on design problem solving has shown that providing designers with representational tools and concrete metaphors can facilitate design work (Carroll, Thomas, and Malhotra, 1980). The problem is that outside the laboratory, when problems cannot be tailor-made to exemplify particular properties, effective representational tools and metaphors cannot always be identified a priori. Providing designers with structuring tools that assist them in achieving a stable problem decomposition can aid the design process (Carroll, Thomas, Miller, and Friedman, 1980). But again, outside the laboratory (Alexander and Simon notwithstanding), it is not always possible to predict the best problem decomposition, and hence not always possible to provide appropriate structuring tools. Nevertheless, we may certainly be able to understand design problem solving better than we now do, and support it much of the time.

We do not believe that this will be as straightforward and simple a matter as do Simon and Alexander, however. In particular, the fact that design is often radically transformational and that it incorporates the discovery of further goals renders any purely analytic approach to design inadequate. However, this is not to exclude the possibility that such approaches can be developed as design aids. Indeed, when we view design as a complex problem solving process, we see that analytic methods do have a role to play, as heuristics particularly in the early phases of design. We return to this point later after examining recent analytic approaches to usability objectives in user interface design.

4. Analytic Approaches in User Interface Design.

Recent work on user interface design has often adopted the view of design held by Alexander and Simon. In this approach, specifications for a user interface are given a formal representation. Properties of this representation are then analytically evaluated, to be finally mapped back onto concepts like "easier to use". The argument is made that this method affords a means of evaluating user interface designs *before* the designs are actually implemented.

In examining this work we want to stress several points. First, while the task analysis inherent in these approaches may be useful initially, in providing designers with a representation of relevant concepts and activities, using such a representation as the basis for a usability evaluation of a set of design specifications requires strong psychological assumptions. There is little reason to expect that these assumptions are warranted, and in the few specific examples which have been given, they can be shown to be unwarranted. It would be wonderful if general usability objectives like "easy-to-learn" could be rendered as formal filters on explicitly represented user interface design specifications. But unfortunately this is not now the case, and may never be the case.

Second, the formal evaluation of a given set of design specifications does not provide the kind of detailed qualitative information about learnability, usability, or acceptance that designers need in order to iteratively refine the specifications, rather it assigns a figure (or perhaps a vector) of merit to the design. This could be used to order a set of alternative designs, but contrasting alternative designs is an extremely inefficient means of converging on the best solution. Moreover, as we have seen, real design doesn't work this way.

In a word, analytic approaches do not support the *process* of design. Instead, they beg the question by trying to collapse this process into a computation. In our view, the design process consists of a cycle of iterative definition, assessment and redefinition of specifications, a cycle too unpredictable to be adequately modelled by a computation. The initial definition of design specifications can often rely on analytic methods. But this initial definition is only the beginning of the design process. Unfortunately, this is where the analytic approaches stall.

4.1 Moran's CLG. Moran's (1981) command language grammar (CLG) attempts to provide a general representation for the user interface of interactive computer systems. He would like this representation to provide an analytic definition of *command language* (which he interprets quite broadly, page 36-7), by recursively enumerating all possible command language designs. He also would like the representation to be a serious psychological model of the user's knowledge of the system, as well as of the initial acquisition and routine use of the knowledge. And, apropos of the present discussion, he would like the representation to describe and support design activity in system development: in the generation of designs, in evaluating designs, and in implementing designs. Understandably, the work he actually presents is quite programmatic and incomplete. Unfortunately, his approach seems unlikely to accomplish the third of these three goals (and we must eschew discussion of the first two).

Moran, very much like Simon, is committed to a top-down architecture. The command language grammar decomposes the user interface into a conceptual component, a communication component, and a physical component, which are in turn decomposed into four levels -- task, semantic, syntactic and interaction. This hierarchy eventuates in interface specifications for particular user tasks (such as "reading new mail"), system concepts (such as "mailbox"), syntactic conventions (such as for command contexts), and dialogue structures (such as prompting sequences). Moran is well-aware that real design activity is not organized top-down (p. 44). He regards his proposal as an idealization.

But, as our consideration of Simon and Alexander illustrates, the word "idealization" can be a euphemism in this type of context. For Moran leans on the notion of idealization to propose that his hierarchical levels of representation (task level, semantic level, syntactic level, interaction level) can provide an organization for the decomposition of the design process. His only qualification is an acknowledgement that this top-down decomposition may need to be iterated to reflect bottom-up dependencies. Exactly how this should be accomplished is unclear, as the CLG seems to presuppose a complete and explicit design *before it can generate any representation*. Approaches that view design as a static representation cannot support the design process, but rather must be supported *by it* (see also Jones, 1970: 12 ff.).

Moran's remarks on user error and error recovery make this point from another perspective. The problem is that the command language grammar does not take errors into account in its representation, and hence does not consider user error as a relevant factor in interface design (or in the psychological model of the user). As in the case of his top-down process architecture, Moran acknowledges that the failure of the command language grammar to address error is an idealization (p. 47). He proposes to extend the representation by calculating all potential error sites, and beyond this by identifying those that are the most likely. However, any action site is a potential error site, and the CLG offers no criteria for identifying the most likely sites. Further, the issue should not be conceptualized as merely one of identifying error sites. The availability, or nonavailability, of perspicuous and direct error recovery methods is likely to be of great importance, and issues such as these are well beyond the purview of the command language grammar.

4.2 Reisner's BNF. Moran's command language grammar is difficult to discuss in much depth or concreteness because his presentation of it is extremely general and abstract. From this standpoint, a better example of work on user interface design in the analytic tradition of Simon and Alexander is Reisner's (1983) Backus Normal Form approach. The goals of the approach are more limited, but also more focussed than those of Moran. Reisner wishes to formally analyze the relative complexity of alternative system interface designs, to determine which of two designs is psychologically less complex. The approach is also more concretely developed than Moran's. Reisner has generated specific predictions based on her analyses, and describes empirical work in progress directed at testing them.

Nevertheless, the same general criticisms we raised with regard to Moran apply here too. Reisner's approach also embodies a view of design as a static representation, as evidenced by her focus on overall benchmark contrasts between well-specified design alternatives. Her consideration of user error also is far too limited; she simply posits an increment in probability of error for additional representational complexity. But errors are not probabilities, they are specific events. Evidence that error probability doubled would be of little direct use to designers; they need to know about particular errors, not "error" as an abstract category of user behavior. Knowing that a particular constellation of interface properties entailed a characteristic error or errors could lead to specific improvements in a candidate design.

The foregoing would have to be qualified if there were specific case studies where the analysis did prove useful. And it is to Reisner's credit that she discusses examples in sufficient

detail to assess this. However, the two examples she presents are not convincing. The first contrasts two procedures for issuing a delete command. The point of the example is to show how differences between user groups can be formally analyzed. Her two groups are "naive" and "experienced" users. In her analysis, the only differentiating characteristic between these two groups is that the representation of the naive group's procedure involves a step "retrieve information from external source" where the experienced group has the step "retrieve information from human memory".

The result becomes even less convincing when one considers the content of her contrast between naive and experienced users. An observational study showed that novice users are often quite reluctant to employ external sources, like manuals, referring to their prior knowledge even when they have no relevant prior knowledge (Mack, Lewis, and Carroll, 1983). Conversely, a questionnaire study showed that relatively experienced users like to use manuals (Rosson, 1984).

Reisner's second example is also problematic. Here she contrasts three methods for locating a remote point in a file being edited (use of a locate command, repeatedly scrolling, and calculating the number lines to be scrolled). She equates the "ease-of-use" of a method with its formal complexity in her representation. Repeated scrolling, because its formal description involves the concatenation of more symbols, comes out as more complex, and hence less easy-to-use than the other two methods. However, a monitoring study of experienced users of a text editor showed that these users often choose to use more lengthy methods (Rosson, 1983). Reisner is aware of the limits of a purely keystroke-level notion of complexity. But in trying to go beyond keystroke-level complexity, she invokes very specialized assumptions about cognitive complexity. She posits that the "cognitive action" of copying a number is less complex than that of calculating a number. This may be true in the case she considers, but it is less clear what this approach leads to more generally. To wit, is a logical deduction more or less complex than a calculation?

5. Empirical Approaches to Design.

A complete assessment of the proposals of Moran and Reisner must consider the context within which these proposals are raised. It is important to recognize that until very recently little more than lip service was paid to usability issues in interface design. Design specifications for an interface consisted only of precise descriptions of the sorts of transactions that could occur via the interfacing devices (e.g., screens and keyboards). If one takes this interpretation of design specifications as a given, Moran and Reisner have done quite well. For even in view of our technical critique of their proposals, it must be granted that they have described a systematic role for usability objectives in design.

On the other hand, perhaps we need not accept these givens: why must interface design specifications be limited to input/output transactions per se? Why can't usability objectives be realized in the design specifications themselves, as opposed to being computed from a representation of specifications devoid of usability considerations? Indeed, an alternative to the analytic approach is to take account of usability concerns from the very outset of the design process, and to factor usability into the design specifications as part of the same process that factors functional objectives into design specifications. What's more, and what is quite striking given the presumptions inhering in an analytic approach, this is not a novel suggestion. In many design domains, usability is routinely factored into the design specifications.

A classic presentation of this empirical approach is that of Dreyfus (1953). Dreyfus' book, entitled *Designing for people*, presents case studies of designing ocean liner cabins, commercial aircraft, automobiles, buildings, magazine layouts, in addition to what is probably

his most well-known design project, the standard telephone handset. His initial and general-level design sentiments are very consistent with what Alexander called "unselfconscious" design: "A honest job of design should flow from the inside out, not from the outside in"(page 18). However, his method of determining what this principle means in a given circumstance is totally different from Alexander's formal clustering algorithms.

For Dreyfus, design is a process first and foremost. It is an intimate collaboration between engineers, designers, and clients (page 48). In presenting his various case studies, Dreyfus stresses how much the success of his design work rested on establishing and maintaining these collaborative relationships. Another, and a rather special, party to this collaborative process is potential users (pages 65-7). Dreyfus urges a dynamic focus on users, a focus that iterates throughout the design process, eliminating nonoptimal candidates, suggesting new candidates, and providing grounds for refinement. This focus on users is to be a systematic research focus, not a matter of casual preference checking: Dreyfus specifically takes to task the practice of inviting one's secretary to give the average user's view.

Analytic methods also play a role in Dreyfus' view of the design process. At an early stage of design, Dreyfus stresses the identification of "survival forms", existent design ideas that are relevant and can be incorporated into the current design (page 59). He motivates this in the observation that people will accept something novel if in it they recognize something they already know. A similar line of thinking has motivated work on user interface "metaphors" (Carroll and Thomas, 1982). For example, a word processor is more easily accepted if designed to exploit the typewriter metaphor. After all, a word processor (as something novel) is in many ways like a typewriter (something familiar). The design process, like any process, needs a place to start; it needs to be seeded. Analytic means can suggest reasonable survival forms (e.g., metaphors) as starting points. Indeed, we might extend the notion of survival form to include not only specific existent design concepts, but also general (survival) design principles, such as those countenanced by Moran and Reisner.

But this is only the *beginning* of an iterative process of design cycles. Survival forms (and survival principles) are only starting points in a process that may ultimately refine, alter, or even reject its starting points -- as illustrated in the Carroll, Thomas and Malhotra design dialogue. How does the process move beyond analytic methods? Dreyfus notes that in his wide experience the use of mock-ups has been essential and cost effective (page 62). He discusses an example from a project in which he designed cabins for an ocean liner (page 69-70). Eight staterooms were constructed in a warehouse, to eight different sets of specifications. "Travellers", who had been asked to pack for voyages ranging from one week to three months, arrived to unpack and settle in. Their use of the stateroom facilities allowed the designers to evaluate storage space and the location of lamps, telephones, light switches, chairs, etc. -- without having to make expensive alterations in a finished ship.

But the time-saving and cost-cutting aspects of an empirical design method built around the use of mock-ups and iteration are secondary. The most important aspect of the empirical approach is that it encourages the discovery of design solutions which on purely analytic grounds might have been missed. Dreyfus' book abounds with examples of this. By his own account, he would have never tumbled to the design of the standard telephone handset had he not mocked up hundreds of prototypes in the course of his design work (pages 101-109). Our view is that design activity is essentially empirical. This is not because we "don't know enough yet", but because in a design domain we can never know enough.

A striking example from the user interface design domain is presented by Wright and Bason (1982), who designed two software packages for a "casual" user population. The packages differed in one respect of design philosophy: one package was designed to be maximally consistent with the users' prior knowledge; users were asked how they thought

about their data and what they wanted to be able to do with it, and this formed the basis for the user interface. The second package was also designed with input from potential users, but in this case the designer used this information to determine how the users *ought* to think about their data and operations on it.

On any principled analytic basis the first package should have had the better interface. Its design, after all, had sprung from a survival form -- the prior knowledge of the users. However, the finding was that the second package was better. Although this is a single case study, it suggests a simple lesson: analytic approaches to user interface design, while they might often be helpful, should never be thought of as fully adequate.

The empirical alternative to analytic design is not a novel idea in design science. Dreyfus' book is thirty years old. The approach is not even completely novel in the design domain of system development (see, for example, Bewley, Roberts, Schroit, and Verplank, 1983; Shneiderman, 1983). Nevertheless, system developers sometimes seem to have missed this point (see, for discussion, Gould and Lewis, 1984; Hammond, Jorgensen, Maclean, Barnard, and Long, 1983). And the currency of purely analytic approaches like those of Moran and Reisner further demonstrates that in the user interface domain, the issue is less clearly resolved than it could be. Accordingly, and in the spirit of extending Dreyfus' empirical approach and of more concretely illustrating an alternative to the Moran-Reisner analytic approach, we turn to an example from the user interface design domain.

6. Design, Development, and Usability Specifications.

The proposal outlined in this section embodies the characteristics of design described by Carroll, Thomas and Malhotra (1979) -- a radically transformational, non-hierarchical process that intrinsically involves the discovery of new goals. And because we are concerned here with the design of *user* interfaces, it includes another critical component -- a focus on users throughout the design process. In our view, design of user interfaces begins with the generation of design objectives, followed by a looping process where specifications are generated, behavioral subskills are analyzed, the subskills are tested in a qualitative fashion, and the information obtained is fed back into the design specifications.

6.1 Design objectives. The design of a user interface starts with the intended users. If the users are "casual users", rather than, say, professional programmers, their role in the design process might be very indirect. It might involve on-site interviews or discussion sessions with the designers, rather than actual involvement in design work. The idea is to find out what these potential users do and how they do it -- without prejudging the matter (in the sense of Alexander's "unselfconscious" design). These preliminary user studies should bring to light both the relevant *functional* objectives, what the users need to do (e.g., document creation, calendar management, interpersonal communication), and the relevant *usability* objectives, how people want to work (e.g., they have little time for training, are often interrupted, and don't want to refer to instruction manuals).

The objectives should be fairly abstract and as technology-neutral as possible: "creating documents" is a good functional objective in this sense, but "using a word processor" begs the question. The goal of a functional objective is the discovery of the technology people *need*, not to squeeze them into some preconceived categorization (in the sense of Simon's house designing example). Similarly, the goal of a usability objective is to motivate the discovery of systems providing the needed function in a way that matches the target population's usage needs. The notion of "interruptable training" is an example of such an objective; "modular on-line interactive training", in contrast, presupposes a specific solution to the usage need. Bullen and Bennett (1982) present a detailed discussion of the process of discovering design objectives for office system workstations.

In the typical case, these design objectives will be too general to serve as direct guidance during the design process. They are seeds however for the development of design specifications, the real nitty gritty of design. Most often, the designers must make a best guess at generating an initial set of specifications which seem to meet the objectives. Analytic approaches, such as those described by Moran and Reisner, might be quite useful in this part of the process (as sources of hypotheses and heuristics, not as final-design generators). Ultimately though, the development of a set of design specifications from starting objectives must be a highly iterative process.

6.2 Design Specifications. Just as in the creation of design objectives, the design specifications should include both *functional* and *usability* specifications. Functional specifications codify the function intended for the system -- both the content of the function that is to be provided and its manner of presentation. These specifications should be motivated by both the functional and the usability objectives, with the former motivating the function's substance, and the latter motivating the function's form. For example, the functional objective of "document creation" could combine with the usability objective of "little time for training", to produce a specification for a menu-driven interface for the document creation process. The details of this first set of specifications (e.g., the number and content of the menus, the nature of the input required from the user) will be a function of the intuitions of the designers, available empirical work, and may result at least partially from the analytic problem-solving approaches described in previous sections.

Like the functional specifications, the usability specifications represent a codification of the design plan. However, rather than specifying the *system function* to be supported, these specifications formalize the *user behavior* to be supported. They are also generated through the interaction of functional and usability objectives: the functional objectives point to the behavior that is to be supported, and the usability objectives to the level at which it is to be supported. To continue with the example above, the "document creation" and "little time for training" objectives might lead to a usability specification codifying the amount of time necessary for a naive learner to create his or her first document. Again, the details of the specification will be a function of intuitive, empirical, or analytic work on the part of the designers, as they attempt to fully specify a behavioral target.

A critical feature of a usability specification is that it be testable. A specification can be evaluated and modified during design only if successes and failures in meeting are patently clear. This means that the satificing behavior must be clearly stated. Further, if a usability specification is to be meaningful, it must reflect an understanding of the target audience and their needs; the specification must make reference to representative users performing representative tasks.

At this point, the reader may wonder if we are proposing anything other a top-down process of design that includes an increased focus on users' requirements. But what we have described so far -- the initial search for design objectives and the creation of specifications -- is only the first step. As it is the first step, it is a critical one: the design process that follows will be greatly influenced by these initial ideas. The essence of our proposal, however, lies in the understanding that design is a temporally extended and highly interactive process, not a problem to be solved once and for all.

6.3 Subskill requirements. The active process we are imagining is schematized in Figure 1. Here, we can see the part of the process described already, in which an analysis of users' functional and behavioral needs leads to objectives and initial design specifications. But note that these specifications comprise only the first state of a design loop. Designers must generate their initial specifications only with the recognition that they may be re-specified or discarded as the process proceeds.

Figure 1 About Here

The remainder of the loop consists of making explicit and testing the behavioral *subskills* implicit in the functional and usability specifications. Given a usability specification for naive users learning to create menus with a menu-driven word processor, the design team must determine the behavioral subskills implied by fulfillment of the usability specification. For example, one subskill required for successful learning of such a system is the comprehension of menu content (the choices listed and the prompts) by naive users; if users cannot understand the displays presented to them, they are unlikely to interact with them successfully at all, much less in the small amount of time targeted by the usability specification.

These subskill requirements need not be phrased in criterion form, as their purpose is not to make a guess at some a priori behavioral goal, but rather to provide for early and continuous usability testing of interface components. While they must be stated in *testable* form, their focus should be on gathering information that suggests specific design solutions to specific interface problems. As for the usability specifications, the subskills identified must be tested for representative users. It would mean little if a sophisticated programmer were able to understand the information presented on a given display panel of a word processing system intended for a secretarial population. Similarly, discovering that temporary office help have difficulty paraphrasing options for a system designed to produce documentation for programs would be of little use.

Most importantly, though, the subskill testing must produce results that have direct implications for design. Discovering that on the average, representative users can correctly paraphrase only half of the choices on a menu will not be very useful to the menu designer. The test then becomes more like that proposed for usability specifications, providing a quantitative picture of the usability of this interface component, but little guidance toward improving it. Instead, the results of subskill testing should be categorized and described in detail. Perhaps, for example, only options based on ranges of numbers fail to be understood; such a description indicates to the designer a probable source of the problem. Indeed, if no useful classification can be found, then problems should be described individually, as this preserves as much of the information they contain as possible.

Notice that in proposing the analysis and testing of subskill requirements, we are incorporating a decomposition process into our approach. However, our use of decompositional processes differs sharply from those of the analytic approaches described in the earlier sections of the chapter. Our decomposition is intended only to support the iterative user testing necessary for successful design, not to replace it. Further, because in our view, design specifications are generated iteratively, so are the subskill requirements implied by them; the decomposition is not a one-shot procedure. For example, testing error recovery subskills might lead to the conclusion that a special "training wheels" system that blocks users' errors is the only way to satisfy the requirement. This result would cause changes in both the functional specifications (they would now include an additional "training" interface) and the usability specifications (the initial learning task would be performed using this special mode); these in turn would lead to changes in the subskill requirements (the naive user now need only recognize that an error has been made, as the training system removes the need for recovery).

Operationalizing the subskills implicit in the design specifications serves two important purposes in the design cycle. First, it supports *early* user testing. Although the usability specifications themselves are certainly testable, they refer to tasks representative of those to be performed on the final system. As a result, they are contingent on the availability of a working prototype, and assume conditions appropriate for gathering quantitative performance

measures. In contrast, the subskill analysis is meant to support the testing of pieces of the system (e.g., menus), as soon as they are available.

A second function of the subskills analysis is the provision of an explicit investigatory tool for the design process. The goal of the analysis is not to "measure" usability, but rather to explore it. As such, the detail is critical, as only through identifying specific problems or successes with a developing interface can a design team obtain the information it needs to support further design.

7. An Example: Designing a Learnable Word Processor.

Imagine a design team faced with the problem of designing a word processing system intended for a secretarial population. One of their key usability objectives would probably be that the system require little training. Given this objective, and given general guidelines relevant to the learnability of user interfaces, several early decisions might be made. For example, the designers might decide to use a menu-based interface, as this allows for *selection* rather than *generation* of the steps necessary to complete a task. Further, let us imagine that the learnability analysis prompted a decision to incorporate contextual dependence in the menu options displayed. Studies of novices learning current systems indicate that a great deal of time is spent recovering from inappropriate menu choices (Carroll & Carrithers, 1984). Presentation of only the possibilities that are appropriate to the user's current state should remove this class of errors.

With these learnability-based interface decisions in mind (of course this example is for illustrative purposes -- in an actual design situation, the number of interface principles would be considerably larger), the designers would attempt to build a system meeting the system's functional objectives (e.g., document creation, revision, formatting, printing). If they were indeed able to generate a design that met both sets of requirements, they might feel that they had been successful in designing a "learnable" system, and proceed blythely to implement the system and put it through its final "assurance" testing. But if this were all they did, they would undoubtedly suffer a rude awakening.

Users' reactions to a proposed design solution simply cannot be predicted (recall the findings of Wright & Bason, 1982). As a result, incorporating user testing after a design has been implemented will always be too late. The complex interplay among specifications during design may promote the discovery of entirely new goals for meeting these needs. Even if the designer's response to usability needs (e.g., developing a menu-based interface) are basically sound, seemingly small details of the interface implementation can have severe consequences for its ultimate usability. We have observed cases, for example, where the wording of a single menu prompt caused users to be stymied for long periods of time in learning to use a system (Mack, Lewis, and Carroll, 1983). If, however, the design process includes user testing from the very beginning, subtle interface "tuning" problems such as these, as well as insights about more radical transformations, will be discovered.

To continue with our example, let us return the design team to the first step of the process. Usability objectives have been stated and have prompted general interface decisions. It is at this point that interface specifications are generated. Typically, such specifications are simply functional -- they describe how the system's function will be made available to the user (e.g., specifying the sequence and content of menus, the logical steps needed for reaching some state, the function available at any point). In our proposal, though, usability specifications and subskill requirements are generated concurrently.

7.1 The functional specifications. In our example, we will focus on the creation of documents (see Figure 2). Our imagined functional specifications for this process indicate that

the user begins by selecting a CREATE NEW DOCUMENT option from the TASKS AVAILABLE menu. The selection is made by pointing at an option with a mouse and pressing the select button. Because of the contextual dependency principle, the other selections displayed on this menu will vary as a function of prior work-- PRINT DOCUMENT ON PAPER will only appear if a printable document exists, for instance.

Figure 2 About Here

Once the creation task has been initiated, the user is shown another menu (CONTROLLING DOCUMENT FORMAT) which offers a chance to set various formatting parameters (e.g., tab settings, margins, line spacing), or to accept the system's default values. As on other menus, decisions are made by selecting a given option (e.g., CHANGE TAB SETTINGS) with the mouse, which brings forth an ancillary menu asking for new parameter values to be typed in over the defaults presented. The content of the DOCUMENT FORMAT menu and its ancillary panels will always be the same when setting up a create job, as there are no contextual constraints on initial formatting decisions.

After making these formatting decisions, the user is presented with an area into which text can be typed. Also available are formatting options which can be embedded within the document. Some of these options are exercised directly with function keys (centering, underscoring). Others (changing tabs, margins or line spacing for a text segment) are accessed by pressing a "FORMAT" key which causes the "CONTROLLING DOCUMENT FORMAT" menu to be overlaid on the screen, as far away from the current cursor position as possible. Note that the selections available from this menu will vary as a function of context, as not all format changes are possible from all states. So, for example, it is inappropriate to change line spacing in the middle of a line of text. Similarly, the "USE STANDARD SETTINGS" option will only appear if some of the format settings have been changed initially.

At any point, the user may "complete" the document by using a SAVE function, or may return to previous states by using the CANCEL function.

7.2 The usability specifications. The design team's key usability objective, "little time for training", must next be operationalized as a usability specification. This requires that the designers be aware of the target population (or populations) and the tasks that will be done using the system by these users. In developing a word processing system intended for secretaries, it would be inappropriate to specify criteria for professionals keying in and editing their own work. Rather, the specifications should refer to office workers, and the reference task should represent one most typical of that population (e.g., keying in or editing memos written by principals). After identifying the target users and their representative tasks, the next step is the establishment of a best guess at a performance criterion -- a good place to start is with an analysis of users' performance on existent systems. To continue with our focus on the user, though, another important input to these criteria might be the target users' views on the kind of performance that would satisfy their needs and expectations of the system.

Studies of novices learning to use word processors have indicated that they want to produce a work product as soon as possible, not simply "learn function" (Carroll, Mack, Lewis, Grischkowsky, and Robertson, 1984). In a survey of users of text-editors, the most common work product produced by secretaries was memos or short letters (Rosson, 1984). Thus, a reasonable criterial specification for our example system might be:

> 90% of a secretarial sample with no prior word processing experience, using only the training materials provided, will be able to create and print a memo in 30 minutes.

This specification addresses the learning of very basic function on the system. And while this is certainly a critical measure of the "little time for training" objective, the specification process should not stop here. Also specified should be criteria for learning more advanced function. A specification to this end might be:

> After successfully creating and printing a memo, 90% of a sample of secretaries with no word processing experience, using only the training materials provided, will be able to create a two-page report with an embedded table in 40 minutes.

In our proposal, these usability specifications provide the behavioral goals the system will strive to meet, as well as ultimately serving as a concrete record of the product's usability. But in addition to setting these behavioral goals, the design team must begin the analysis and testing of behavioral subskills that will guide them through the design process.

7.3 The subskill requirements. There are several classes of subskill requirements implicit in learning to create documents in our imaginary system. At each step of the process, novices must be able to understand any information provided, recognize what responses are possible, produce appropriate responses, and recover from errors encountered in making these responses. As for criterial specifications, these subskills are stated in a testable form by indicating the referent user population and the satisficing behavior. To continue with our example, the following four subskill requirements would be included in the "create document" portion of the design specifications, directly after the usability specifications outlined earlier:

> 1) Any printed or displayed material encountered in the process of creating a document will be understood by secretaries with no prior computer experience, as demonstrated by their ability to paraphrase the content.
>
> 2) The response expected from the system at any given time will be obvious to secretaries with no prior computer experience, as evidenced by their ability to describe or execute a response that would take them to the next step of a given scenario.
>
> 3) The procedures for proceeding through the states necessary to create a document will be executable by secretaries with no prior computer experience, as evidenced by their ability to carry out instructions describing these procedures.
>
> 4) The system will provide error recovery mechanisms suitable for secretaries with no prior computer experience, as evidenced by their ability to describe or execute successful recovery strategies given an error scenario.

These specifications refer to a population (computer-naive secretaries), and indicate both what the subskill is (e.g., understanding the information presented), and what observable behavior will be used to study users' success with the subskill (e.g., paraphrasing the information). More importantly, though, they refer to components of the full system that can be tested early in the process, and whose testing will yield the detailed information needed for continuing the design process.

7.3 The iterative design process. The usability specifications and subskill requirements are intended to engender a considerable amount of testing during the process of design. Indeed, it may seem that the amount of testing implied at this point is enormous. Not only are we assuming that performance on representative tasks be carefully measured at some point, but also that display panels, training materials, expected responses, and error recovery abilities be explored prior to these performance tests.

We propose that the solution to the testing problem is to focus not on conducting a large number of experiments over a large number of users, but rather on extracting as much information as possible from every user who comes into contact with any part of the system. And we feel that the way to do this is by adopting a more informal, qualitative approach to testing. Much can be learned, for example, in close observation of a single novice using a word processor for the first time, particularly if the user is asked to "think aloud" while working.

These suggestions can be made more concrete by returning to our example. A key component of the testing, of course, is that it begin *early* in the design cycle. As soon as there is any aspect of the interface that can be exposed to users, testing should begin. A good example comes from the subskill requirement that printed and displayed material be comprehensible by naive users.

Early testing of this subskill can begin by mocking up the first attempts at display panels on paper. An example of the type of testing we have in mind might proceed as follows: Several secretaries are given the task of paraphrasing the choices listed on the panels relevant to creating documents; they are also asked to think aloud as they perform the task. One user might be observed to have difficulty with the phrase "SET MARGINS", commenting that one of the other phrases, "USE CURRENT SETTINGS", seemed to suggest that margins were already set-- so why would they need to be set again? Such comments provide valuable information to the panel designer, as they point to subtle interactions among the phrases chosen for options within a panel. Indeed, the user's comment suggests a solution to the comprehension difficulty-- make the distinction between pre-assigned values and new assignments more obvious, as in "ACCEPT ALL CURRENT SETTINGS" and "CHANGE MARGINS SETTINGS".

Early interface testing need not be confined to paper and pencil tasks. So, for instance, the process of moving through a menu tree could be simulated quite easily in advance of a working prototype. As soon as a set of potential menus is available, menu scenarios can be set up to test users' ability to recognize and execute appropriate responses at each step. Here again, the thinking aloud methodology combined with close observation of the users will allow designers to gather a great deal of information from relatively few users. One might discover at this stage that while novice users understand the request to type in a document name, they have difficulty generating document names (cf. Carroll, 1984). Such an observation could lead to an insight about unique system-provided names (e.g., "Document n") which the user can simply accept if desired.

Designers might use a similar simulation technique to examine users' ability to recover from errors. Although some errors will probably not occur until a fully functional prototype can be used, common errors in menu interactions can be detected. To get as much information from as few users as possible, error conditions could be simulated: Instead of a direct path to the typing area for simple memo creation, a user could be shown an inappropriate venture into some of the document formatting menus, and then asked for ways to "get" to the desired goal. By careful probing, the design team might discover not only whether the mechanisms available (e.g. error correct backspace, cancel) were sufficient, but also what kinds of mechanisms these users would *like* to have (e.g., stop and start all over).

Work in this low-cost simulation environment would allow easy revision of menu content, or even of menu structure, in response to user problems. However, it is also critical that working prototypes be available as early as possible, as many problems can be expected to occur only when the user is required to deal with the full complexity of the system. Of course, the fact that the components of an interface are combined into a working prototype should not be taken as a sign that the detailed observations made early in the design process should be

dropped in favor of more global performance measures. To the contrary, detailed observation is even more critical in this stage, as use of the system becomes more demanding, and as the interface nears its final stages of development.

Although testing with a full prototype may be possible only relatively late in the design process, there is no guarantee that problems discovered at this point will be minor "tuning" issues. Consider the following example: A user is working with a prototype system, and is entering text in the typing area. At one point, the user needs to create a small table, and to do this, must set new tab settings, and change the line spacing. The user correctly presses the FORMAT key, and the CONTROLLING DOCUMENT FORMAT menu appears on the side of the screen. However, quite unexpectedly, the CHANGE TAB SETTINGS and CHANGE LINE SPACING options are not there. This is due to the fact that the menu was requested while the cursor was located in the middle of a line, an inappropriate place to make tab and line spacing modifications: the system is "protecting" the user from making mistakes. In this case, though, the protection may have disastrous effects. A typical reaction might be to accept that "Oh, I guess I get a different FORMAT menu once I start typing; there must be a different way to make these changes". Clearly, such an account could have long-lasting consequences for the learner's understanding of embedded format settings.

This scenario provides an example of the non-hierarchical nature of design. Initially, the concept of contextual dependency seems sound, and indeed there are probably numerous occasions in which it saves novices from entering mind-boggling error states. But in any particular case, application of this top-level directive may be exactly the wrong thing to do.

How might this design problem be resolved? One might opt for an heterarchical solution: disregard the dependency principle for this special case. Continue to make display of choices in other menus contextually dependent, but always display the full complement of format menu choices. The risk here, of course, is exactly that which the principle seeks to minimize--the user is more likely to select inappropriate choices.

Another solution embodies a new operationalization of contextual dependency. Instead of making the *display* of options contextually dependent, the dependency could be incorporated into the *selection* process. In our problem case, the user now would see the CHANGE TAB SETTINGS and CHANGE LINE SPACING options. Indeed, the menu would look exactly as it did on any other occasion. But these options would not be selectable. While this could still leave the user confused as to what the exact problem was, it would at least reinforce the belief that this was the appropriate menu to interact with.

To this point, we have focussed on the exploratory testing of subskill specifications. This is a natural consequence of an emphasis on the *process* of design. Indeed, as we suggested earlier, one consequence of the exploratory testing of subskills might be substantial revision of the usability specifications. Perhaps, for example, early testing of the training materials indicates that it will be very difficult for novices to use them unassisted. This may induce the incorporation of a training "hotline" into the resources provided for the learner, and the usablility specifications would have to be modified to reflect this change.

As early as possible, quantitative tests of the usability specifications must also be conducted. This provides yet another reason for creating working prototypes of the system and training materials as early as possible. Although the exploratory, qualitative testing of subskills is crucial, it will never be enough. There is no guarantee that competency on the various subskills will combine to meet the performance specifications. Indeed, it may be that the system will never meet the specifications (or alternatively, far surpass them). Nonetheless, it is important to record representative users' performance on representative tasks, as this serves as a metric for describing the final usability of the system.

In this example, we have emphasized both the process nature of interface design, and its inherent tuning and rejection of interface specifications. Indeed the problem has been minimized to an extent by the focus on specifications relevant to a system's learnability. In real world design, the number and types of usability specifications will be much larger, as concerns for installation, experienced use, and maintenance are figured in. Moreover, the functional objectives and their codification as functional specifications cannot be presupposed, as in our word processor example. Rather, empirical studies of what people *need* must dictate the functional and usability objectives for new applications. These in turn must be codified as specifications, and the subskills implicit in them recognized and tested in an iterative process.

8. A Final Word.

Design will always be a hybrid discipline. It is both science and art; it is both formal and empirical (Jones, 1970). Recent considerations of user interface design -- and of design theory more generally -- have focused inordinate attention on formal aspects of design at the expense of adequate attention to empirical concerns. This is an unfortunate imbalance in that the empirical aspect of design, the part least amenable to formal analysis, is also the most critical: Design is a *process,* it is not a state and cannot be adequately represented statically. The design process is *non-hierarchical,* neither strictly bottom-up nor strictly top-down. The process is *radically transformational,* involving the development of partial and interim solutions which may ultimately play no role in the final design. Design intrinsically involves the *discovery of new goals.*

We have stressed the empirical aspect of design in our characterization of usability specifications and their associated behavioral subskills. The decomposition of usability objectives into specifications, and finally into subskill requirements, is an iterative empirically driven process -- a process of refining and rejecting interim solutions, and of discovering new solutions. This design loop of iterative decomposition, refinement, and redefinition is at the heart of our view of the design process. Design is a developmental process, in the sense used by Whiteside and Wixon (this volume), not a mechanistic process.

This view of design has particular implications for the role of behavioral expertise in user interface design. For if we think of design as a radical process of discovery and redefinition, then it simply won't do to think of behavioral expertise as something that can be brought in at particular points and stages of a design plan. We must think of behavioral work as part of the design process, as intrinsic to it. Useful and usable systems can only be designed deliberately.

Acknowledgements.

Our critique of analytic approaches to design owes much to an earlier collaboration between the first author and John Thomas and Ashok Malhotra (circa 1977). Our subskill analysis of usability specifications was originally developed by the second author in a talk at the IBM Austin Laboratory in September 1982. A briefer version of this chapter appeared in a special issue of BYTE Magazine (Carroll and Rosson, 1984). We are grateful to Phil Hayes, Phyllis Reisner, Ben Shneiderman, John Thomas, John Whiteside, and Dennis Wixon for commenting on this chapter. We are especially grateful to Clayton Lewis for patient and incisive comments on several earlier versions of this chapter.

References.

Alexander, C. *Notes on the synthesis of form.* Cambridge, MA: Harvard University Press, 1964.

Bewley, W.L., Roberts, T.L., Schroit, D., and Verplank, W.L. Human factors testing in the design of Xerox's 8010 "Star" Office Workstation. *CHI'83 Conference Proceedings, SIGCHI Bulletin,* December 1983, 72-77.

Bullen, C.V. and Bennett, J.L. Office workstation use by administrative managers and professionals. IBM Research Report, RJ 3809, 1983.

Carroll, J.M. Designing MINIMALIST training materials. IBM Research Report, 1984.

Carroll, J.M. and Carrithers, C. Training wheels in a user interface. *Communications of the ACM,* 1984.

Carroll, J.M. and Mack, R.L. Learning to use a word processor: By doing, by thinking, and by knowing. In J.C. Thomas and M. Schneider, (Eds.) *Human factors in computing systems.* Norwood, NJ: ABLEX Publishing, 1983.

Carroll, J.M., Mack, R.L., Lewis, C.H., Grischkowsky, N.L., and Robertson, S.R. Learning to use a word processor by Guided Exploration. IBM Research Report, 1984.

Carroll, J.M. and Rosson, M.B. Beyond MIPS: Performance is not quality. *BYTE,* 1984, *9/2,* 168-172.

Carroll, J.M. and Thomas, J.C. Metaphor and the cognitive representation of computing systems. *IEEE Transactions on Systems, Man, and Cybernetics,* 1982, *12,* 107-116.

Carroll, J.M., Thomas, J.C., and Malhotra, A. Clinical-experimental analysis of design problem solving. *Design Studies,* 1979, *1,* 84-92.

Carroll, J.M., Thomas, J.C., and Malhotra, A. Presentation and representation in design problem solving. *British Journal of Psychology,* 1980, *71,* 143-153.

Carroll, J.M., Thomas, J.C., Miller, L.A., and Friedman, H.P. Aspects of solution structure in design problem solving. *American Journal of Psychology,* 1980, *93,* 269-284.

Dreyfus, H. *Designing for people.* New York: Simon and Schuster, 1955.

Gould, J.D. and Lewis, C.H. Designing for usability -- key principles and what designers think. IBM Research Report, 1984.

Hammond, N., Jorgensen, A., Maclean, A., Barnard, P., and Long, J. Design practice and interface usability: Evidence from interviews and designers. IBM Hursley Human Factors Laboratory Report, HF 082, 1983.

Jones, J.C. *Design methods.* New York: Wiley-Interscience, 1970.

Kidder, J.T. *The soul of a new machine.* New York: Avon Books, 1982.

Mack, R.L., Lewis, C.H., and Carroll, J.M. Learning to use word processors: problems and prospects. *ACM Transactions on Office Information Systems,* 1983, *1,* 254-271.

Malhotra, A., Thomas, J.C., Carroll, J.M., and Miller, L.A. Cognitive processes in design. *International Journal of Man-Machine Studies,* 1980, *12,* 119-140.

Moran, T.P. The command language grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies,* 1981, *15,* 3-50.

Reisner, P. Formal grammar as a tool for analyzing ease of use: some fundamental concepts. In J.C. Thomas and M. Schneider (Eds.) *Human factors in computing systems.* Norwood, NJ: ABLEX Publishing, 1983.

Rosson, M.B. Patterns of experience in text editing. CHI '83 Conference on Human Factors in Computing Systems, Proceedings. Boston, MA: December 12-15, 1983.

Rosson, M.B. Effects of experience on learning, using, and evaluating a text editor. IBM Research Report, RC 10322, 1984.

Shneiderman, B. Human factors of interactive software. In A. Blaser and M. Zoeppritz (Eds.) *Enduser systems and their human factors.* New York: Springer-Verlag, 1983.

Simon, H.A. The structure of ill-structured problems. *Artificial Intelligence,* 1974, *4,* 181-201.

Wright, P. and Bason, G. Detour routes to usability: a comparison of alternative approaches to multipurpose software design. *International Journal of Man-Machine Studies,* 1982, *18,* 391-400.

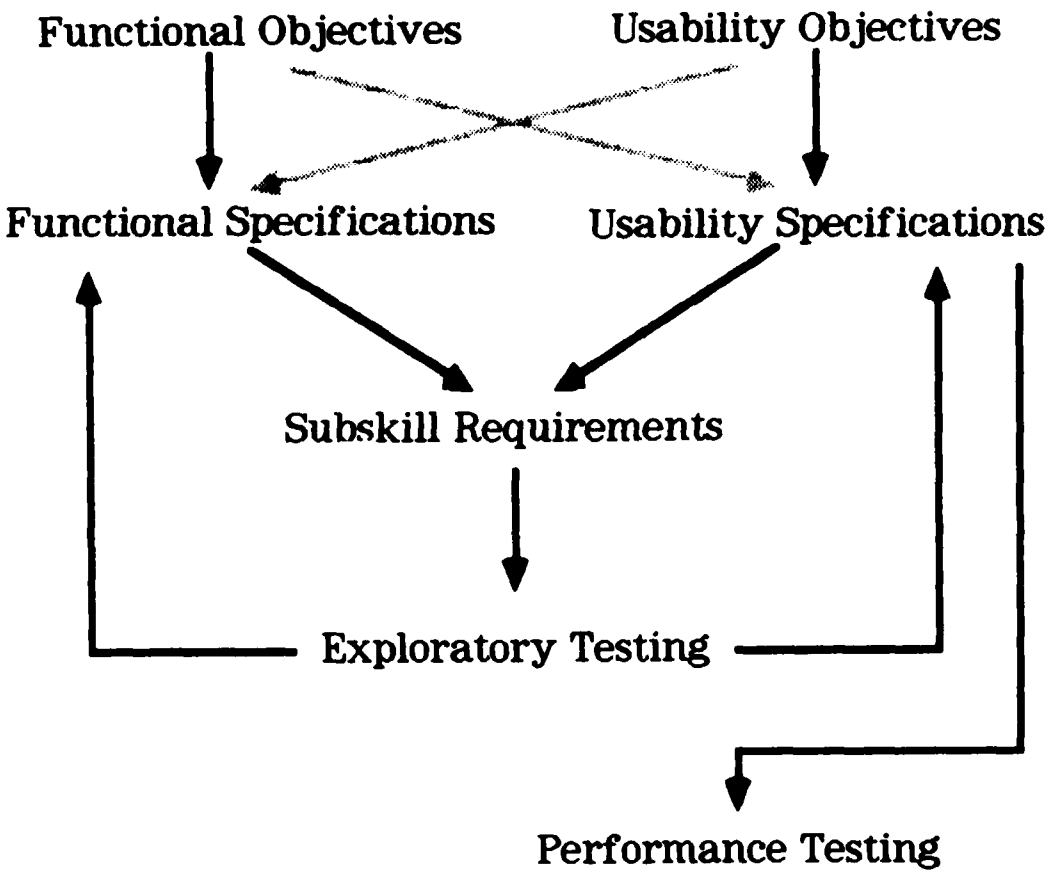


Figure 1: *Schematic representation of the iterative design process.*

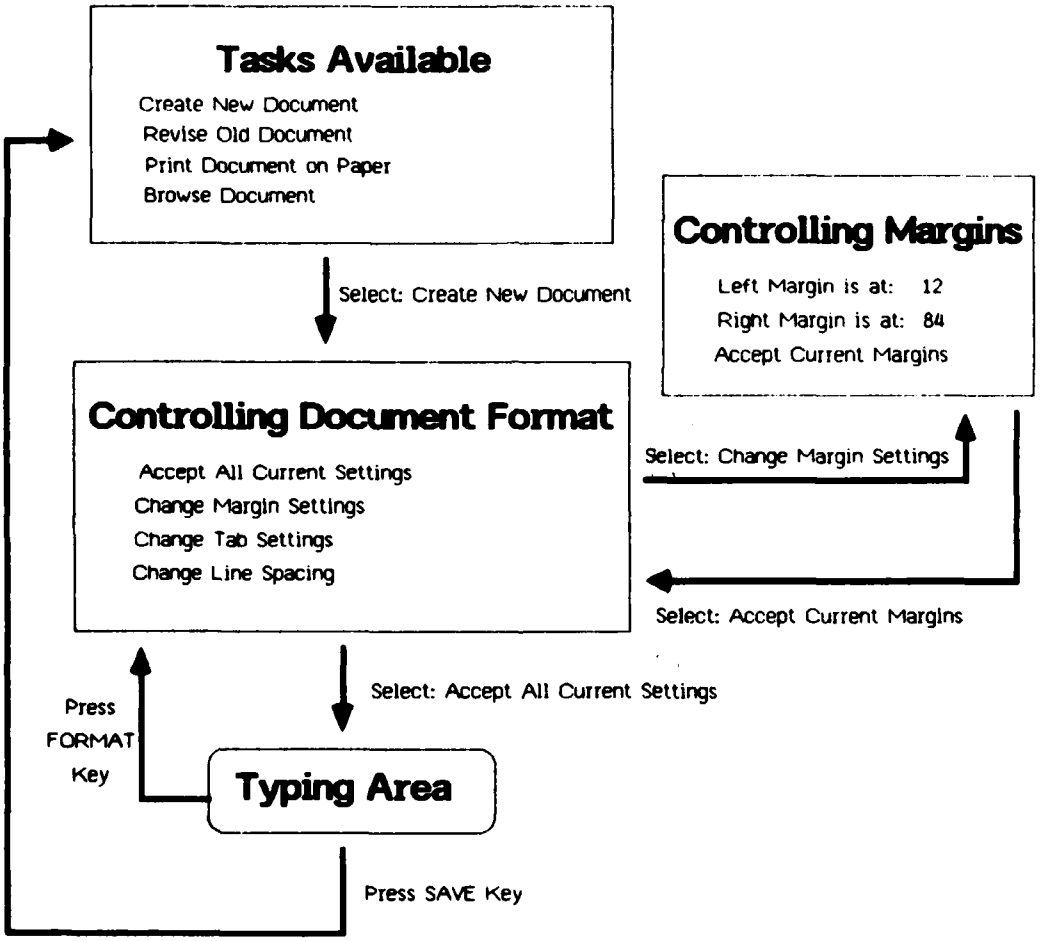


Figure 2: *Control structure for creating a document in example system.*

END

FILMED

3-85

DTIC